Complex Systems **3** (1989) 229–241

Learning by Minimizing Resources in Neural Networks

Pál Ruján*
Mario Marchand†
Institut für Festkörperforschung der Kernforschungsanlage, Jülich, Postfach 1913, D-5170 Jülich, Federal Republic of Germany

Abstract. We reformulate the problem of supervised learning in neural nets to include the search for a network with minimal resources. The information processing in feedforward networks is described in geometrical terms as the partitioning of the space of possible input configurations by hyperplanes corresponding to hidden units. *Regular partitionings* introduced here are a special class of partitionings. Corresponding architectures can represent any Boolean function using a single layer of hidden units whose number depends on the specific symmetries of the function. Accordingly, a new class of plane-cutting algorithms is proposed that construct in polynomial time a "custom-made" architecture implementing the desired set of input/ouput examples. We report the results of our experiments on the storage and rule-extraction abilities of three-layer perceptrons synthetized by a simple greedy algorithm. As expected, simple neuronal structures with good generalization properties emerge only for a strongly correlated set of examples.

1. Introduction

Connectionist models of cognition attempt to give a microscopic description of how ensembles of neuron-like units process distributed information in parallel [1]. Simple processing units mimicking neurons are connected into a multilayered network with *visible* input and output layers and one or more internal layers of *hidden units*. The presence of hidden units is essential if the network is to perform complicated tasks, since *perceptron-like* devices without hidden units have only limited abilities [2]. These feedforward networks

*On leave from Institute for Theoretical Physics, Eötvös University, Budapest, Hungary. Electronic mail address (Earn/bitnet): IFF162@DJUKFA11.

†Present address: Department of Physics, University of Ottawa, 34 G. Glinski, Ottawa, Canada K1N-6N5. Electronic mail address (Earn/bitnet): MMMSJ@UOTTAWA.

are especially useful because their simple dynamics allows for the recursive calculation of the error gradient by the backpropagation method [3].

In these models, learning is considered mainly as a *memorization-in-a-network* problem: given a task (i.e., a set of input-output pattern pairs) and a network architecture (i.e., the list of neurons, layers, and their connectivity) one tries, with a given learning algorithm, to find the value of each connection so that the network will perform the task. However, it has recently been shown [4] that the problem of deciding whether or not a given task can be performed by a given architecture is NP-complete. Hence, it is not surprising that there is no known learning algorithm that will answer this *memorization-in-a-network* problem in a number of steps polynomial in the problem size (architecture + examples), and we should not expect one to be found in the near future. It is also not surprising that algorithms like backpropagation are not guaranteed to converge at all and do get stuck quite often in local minima, especially when a network with a minimum number of neurons is used [3,6].

The main objective of connectionist learning is, however, not a bare *memorization* of input-output pairs since this can be accomplished satisfactorily (and in a linear number of steps in the number of examples) by non-neural algorithms like table look-up. Rather, we hope that neural networks will be able to capture the *symmetries* present in the patterns to be learned so that when new, unlearned patterns are processed, the output will be consistent with the observed symmetries. This *rule extraction* ability is required for pattern recognition and inductive generalizations and should be possible only if the task to be performed is "regular," "smooth," or "predictable" in the sense to be explained later. Moreover, it is now recognized [5] that this property can emerge only when a minimal network (i.e., a network with a minimum number of units and connections) is used to learn an illustrative set of examples. Hence, the major problem of connectionnist learning (within the supervised learning paradigm) is trying to *find* a minimal network performing a given task rather then trying to *load* [4] a given task into a given architecture.

Unfortunately, the known learning algorithms are defined for a given, fixed architecture and thus are not suited to find a minimal network. Indeed, they attempt only to answer the *memorization-in-a-network* problem. It is thus meaningless to pretend that such *algorithms* like backpropagation have generalization or rule-extraction ability since it is the *human* designer who must guess the minimal architecture that will make generalization possible. Moreover, because of NP-completeness, it is computationally intractable to try to find a minimal network by using these learning algorithms. When such a strategy is adopted, say with backpropagation, one is confronted immediately with the practical problem of deciding what to do when stuck in a local minimum: should we try to find a lower minimum that correspond to a solution? Should we add more neurons and connections (if we suspect that this tentative architecture cannot perform the task)? Obviously, learning procedures that aim at constructing a minimal network (without fixing

the architecture) implementing a given task are badly needed. This paper presents our results of such an attempt.

Before summarizing our results, let us make a few remarks on algorithmic complexity. First, finding a network with the minimum number of neurons and connections that perform a given task is obviously at least as difficult as trying to load that task into a given architecture. We are not interested, however, in finding the absolute minimum but only one that give us sufficiently good generalization abilities. How far should we go? This is perhaps the most important question in supervised learning theory. In fact, it is very easy to *find* a three-layer network that will perform any possible given task [5]: one only needs to *assign* one neuron for each pattern of the task — the incoming connections to each hidden neuron being determined so that it fires only in the presence of that particular pattern (see the Grand-Mother solution in the next section). This local solution, obtained in only one pass through all the patterns, has no generalization ability since the output of the net on the unlearned patterns is not correlated to the output of the net on the learned patterns. Reducing the number of hidden units will build these correlations and hence make generalization possible. We therefore conjecture that, given any "regular," "smooth," or "predictible" task, it is possible to construct a network, in a *polynomial* number of steps, that will have less neurons than the above solution and will possess some generalization ability.

To prove this statement, we present in this paper a new class of learning procedures that aim at constructing a minimal neural network corresponding to the desired task. Using the notion of *regular partitionings*, we show that a simple "greedy" algorithm always finds in *polynomial* time[1] a solution regardless of the regularity of the task to be learned. When the task consists of uncorrelated patterns, networks with many hidden units and no generalization ability are found. On the contrary, if the task consists of strongly correlated patterns, networks with few hidden units and good generalization ability are found as solutions.

2. The concept of regular partitioning

In order to obtain a learning procedure which constructs the minimal network for a given task, we have to limit ourselves to certain types of architectures and neuron-like units; otherwise, the search space becomes unnecessarily wide. For instance, we consider here only feedforward layered neural networks since their dynamics is simple but capable of universal computation. An example is shown in figure 1a. The network consists of simple logical threshold McCulloch–Pitts units connected to each other by inhibitory or excitatory connections. Such an unit is shown in figure 1b. The output is

[1]By this we mean that the number of elementary computational steps (like arithmetic operations, comparisons, assignments...) that are necessary to find a solution is increasing polynomialy with respect to the size of the instance of our problem which is the number of bits needed to store the patterns to be learned and not only the number of input units of the network — as often incorrectly used.

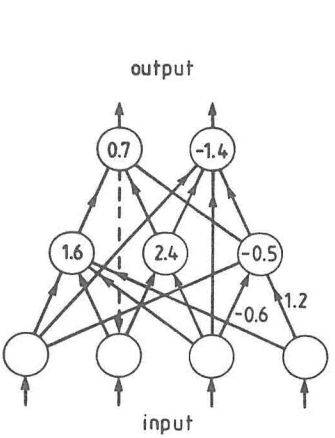

Fig.1a

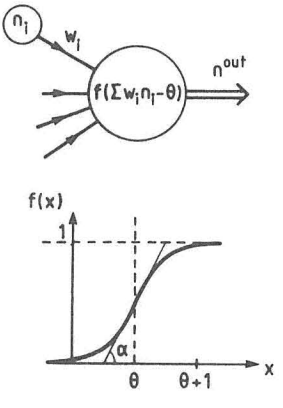

Fig. 1b

Figure 1: A typical feedforward architecture. The information is processed from bottom up; feedback loops as the one shown by the broken line are not allowed. The nodes are additive threshold units, calculating the output as in equation (2.1). We use the step function obtained in the limit $m = \tan\alpha \to \infty$. Numbers shown on the edges and nodes stand for the connection and the threshold strengths, respectively.

calculated as

$$n^{out} = \lim_{m\to\infty} \frac{1}{2}[1 + \tanh mx], \qquad x = \sum_{\text{all inputs } i} w_i n_i - \Theta \tag{2.1}$$

where the activations are binary variables, $n^{\text{out}}, n_i = \{0, 1\}$. Note that equation (2.1) has a simple geometrical interpretation: n^{out} has value 1 on one side of the plane $\vec{\mathbf{w}} \cdot \vec{\mathbf{n}} = \Theta$ and 0 on the other. The components of the normal vector $\vec{\mathbf{w}}$ are the incident connections (weights) w_i and the threshold of the unit is Θ. The information is processed synchronously and in parallel from the input to the hidden and then to the output layer without the possibility of feedback. While a network with feedback must be iterated until it settles in its stationary or limit-cycle state, feedforward networks process information only until the wave of activity reaches the output units.

Since a network with one layer of hidden units is sufficient in order to execute any Boolean function [5,7,8], we discuss here only architectures of this kind. For the sake of simplicity, we also restrict ourselves to a single output unit (three-layer perceptrons), even though our method is easily generalized to the case of multiple outputs (a more detailed account of our work will appear elsewhere).

Let us first examine closely some geometric properties of this type of networks. In order to make our theory more understandable, we show its main ingredients through the well-known example of the parity-K predicate [2].

The output of the parity-K network should be activated only if the sum of ones in the N_{inp} =K binary input units is odd:

$$n^{out} = \left(\sum_{i=1}^{\mathrm{K}} n_i^{inp} \right) \qquad \text{mod } 2 \tag{2.2}$$

The cube shown in Figure 2 represents the *input configuration space* (or the switching space) for the parity-3 problem. Each corner of the cube corresponds to a possible input configuration, while the color of the vertex shows the desired output value (white = 0, black = 1). In general, for N_{inp} input units, one must consider an N_{inp}-dimensional hypercube. The set of white points shown in figure 2 cannot be separated from the set of black points by a single plane satisfying equation (2.1). The simplest way to see this is to construct the convex hull (the smallest convex polytope incorporating all points in question) of the two set of vertices, S_0 and S_1. In our example, they are two tetrahedrons whose intersection is the octahedron shown by heavy lines in figure 2. On the contrary, when the intersection $M = S_1 \cap S_2$ is empty, the space is linearly separable.

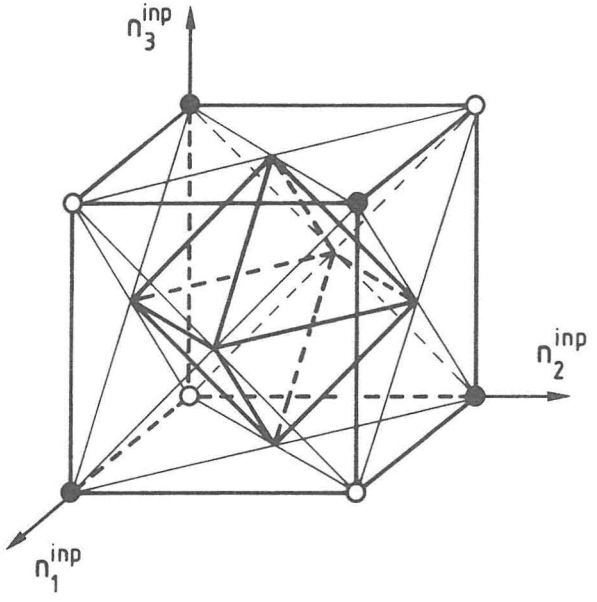

Figure 2: The unit cube representation of the parity-3 problem and the corresponding network. Every corner of the cube corresponds to one possible combination of the binary input units, while the color of the corner shows the desired output value (empty circles = 0, full circles = 1). The two tetrahedrons formed by the white and black set of points intersect on the octahedron displayed in heavy lines.

Consider now the three planes shown in figure 3a. Their normal vector is $(1,1,1)$ and the corresponding threshold values are $1/2$, $3/2$, and $5/2$, respectively. They partition the cube into four disjoint regions, each region containing vertices of the same color. Each plane represents a hidden unit. When running the network over the complete set of inputs, one makes the important observation that the activations of the hidden units form only four distinct configurations, called *internal representations*, out of the possible $2^3 = 8$. This is the *contraction* property: all vertices of the cube contained in the subspaces bounded by the hidden planes are mapped into a *single* internal representation of the hidden (or internal) activation space. Contraction is the natural way through which the network performs classifications and is defined as the number of internal representations over the number of input examples.

Note that such partitioning is by no means unique. Another example is given in figure 3b, where we have simply separated by individual planes all points of black color. Such a partitioning is called a "grandmother" (GM) solution of the problem. The name comes from the theory of local representations, where perception is viewed as filtering the input signals through a variety of feature detectors activating finally a single neuron on the recognition of grandmother. In our picture, this means that every single input set activating the output (black points) has an unique internal representation. Such a partitioning solves always the *memorization* problem but requires an excessively large amount of resources.

The internal (or hidden) configuration space has its own hypercube, whose dimension equals the number of hidden units N_h . We show it for the partitioning of figure 3a in figure 3c. Due to the contraction property, however, only $N_h + 1$ different corners of the hypercube can be colored.

Following these observations, we now define a *regular partitioning* of the unit hypercube in K dimensions as a partitioning with hyperplanes (hidden units) such that:

1. every region contains vertices of the same color,
2. every hyperplane separates corners of different colors, and
3. the planes do not intersect inside the hypercube.

Although regular partitioning is not the only way hidden units may form a separable space (the class of which we call *legal partitionings*), we have not yet found examples of legal partitionings which solve a given problem with less planes than a regular one. Hence, our strategy will be to find the regular partition containing the minimum number of planes.

We now show that every regular partitioning is separable by a single output unit (hyperplane of the hidden space). The proof is quite simple: imagine a point inside every region, representing a given activation pattern of the hidden units. Construct a graph connecting all nearest-neighbor points (see figures 3d and 3e). The N_h edges of this graph must pass through a single plane delimiting that region and no other edge can cross this plane (from property 3 and convexity). All nodes of the graph must have neighboring

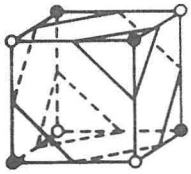

Fig. 3a

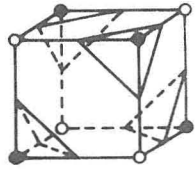

Fig. 3b

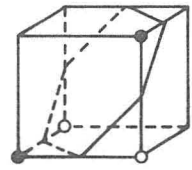

Fig. 3c

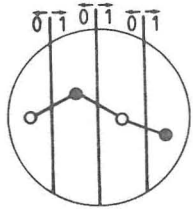

Fig. 3d

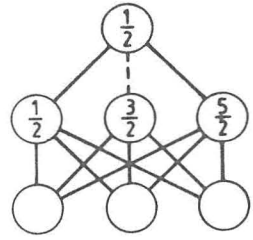

Fig. 3f

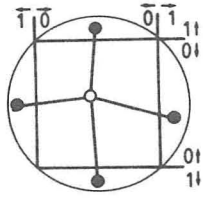

Fig. 3e

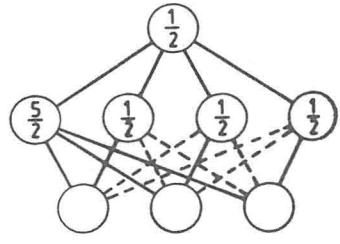

Fig. 3g

Figure 3: Processing information in feedforward networks. Figure 3a shows a partitioning with parallel (1,1,1) planes, while Figure 3b is a "grandmother-type" partitioning. Figure 3c is the configuration cube of the hidden units (internal layer). We use the same notation as in Figure 3a, except that unmarked corners can never be reached with the chosen allocation of hidden units. The corresponding picture for the grandmother solution involves a four-dimensional cube, not shown here. Figure 3d and 3e are schematic representations of these regular partitionings. The points, each representing a whole cluster of corners, are connected by a bipartite tree (see text). Finally, in figures 3f and 3g, these partitions are expressed as conventional networks solving the parity-3 problem. Full (broken) lines have strength $+1$ (-1).

nodes of the other color (from property 2). Hence, the graph is a bipartite tree (loops are forbidden because of property 3). Each edge points in a different direction because only the corresponding hidden unit changes its activation along that edge. The median points of the edges form a set of non-colinear N_h points, defining uniquely a hyperplane in the N_h-dimensional hidden space. In addition, this plane cannot have 0 connections (every hidden unit must be connected to the output). An example of the parity-3 problem is shown in figure 3c. Expressed in another way, the regular partitionings have the property that when presenting the whole set of examples, the internal representations are linearly independent vectors in the hidden configuration space and are thus linearly separable.

3. The learning procedure

Our goal here is to find the regular partition that contains the minimum number of planes or, conversely, the partitioning with the maximal contraction. Having found that, it is simple to find the set of connections going to the output unit by the procedure outlined in the proof, which amounts to the solution of a set of sparse linear equations. The strategy adopted below is certainly not unique nor optimal but it proves our point.

First imagine that we have at our disposal a greedy algorithm that provides us with the plane that separates the largest cluster of corners having the same color on one of its sides. Cutting out along the plane this set of corners from the hypercube will leave behind a complicated but still convex body. Note that the plane cares only about marked corners corresponding to presented examples. All the other separated but unmarked corners are thus automatically classified as pertaining to the same category, as represented by a specific internal representation. The plane obtained in this way defines our first hidden unit. We can again apply greedy on the remaining convex body but this time we must accept only planes that do not separate the points that have already been excluded by the first plane in order to satisfy property 3 above. We then repeat this procedure until we find a plane separating only points of opposite colors. What we have now is a partition satisfying properties 1 and 3 but not necessarily property 2. However, the planes that separates points of the same color are easily identified and removed (since they are not needed). After disconnecting these spurious units, we have a regular partitioning.

In order to construct the output unit, one has to solve a set of linear equations corresponding to

$$\vec{w}^{out}\vec{m}_{ij} = \Theta^{\text{out}}, \quad \forall \ < i, j > \tag{3.1}$$

where m_{ij} denotes the middle point of the edge connecting the nearest-neighbors vertices i and j in the hidden configuration space. Equation 3.1 can be simplified by observing that when substracting from each other any two equations involving $\vec{m}_{ij}$ and $\vec{m}_{jk}$, respectively, then only two elements of the vector $\vec{w}^{\text{out}}$ will remain. This sparse set of homogeneous equations can

be easily solved by following the structure of the graph connecting nearest-neighbor configurations of hidden units, leaving behind only the (trivial) task of calculating the threshold Θ^{out}.

We now need to define the greedy algorithm. Again, this step, or subroutine, can be done in many ways and we present here only the most straightforward possibility (other variants will be discussed elsewhere). Consider the "minimal set" of hyperplanes, where the connections (weights) may assume only the values $w_i = \{-1, 0, +1\}$ while the thresholds are chosen as $\Theta = -\frac{2N_- - 1}{2}, -\frac{2N_- - 1}{2} + 1, \ldots, \frac{2N_+ - 1}{2}$ where $N_{-(+)}$ counts the number of -1s ($+1$s) in $\vec{\mathbf{w}}$. All such planes intersect the unit hypercube but do not contain any vertex. By a simple counting, we find that their total number is $\mathrm{K}3^{\mathrm{K}-1}$, the number of possible input patterns being $N = 2^{\mathrm{K}}$, so this number can be expressed as $\frac{1}{3} \cdot \log_2 N \cdot N^{\log_2 3}$. This set of planes is nevertheless able to represent any Boolean function (as a grandmother-type regular partitioning, for instance). This means that if the greedy algorithm implements only a simple linear search through all these plane, a minimal regular partition will be found after at most $\frac{1}{6} \cdot \log_2 N \cdot N^{(2+\log_2 3)}$ steps. Of course, faster and more sophisticated searches could also be implemented, but for the scope of this paper we have used only a linear search.

Given a set of examples, this algorithm will construct a three-layer architecture where the units are connected layerwise with all other units through inhibitory and excitatory connections of unit strength (0 corresponds to a missing connection). This seems reasonable from a biological point of view and is also advantageous for digital implementations of such networks. Although one is not guaranteed to find any longer the optimal partitioning with this set of planes, one still expects solutions with good data compression and good generalization properties.

4. Numerical results

To test our learning procedure's capability of dealing with any Boolean function, we have chosen at random 200 Boolean functions defined on 6 bits (among the 2^{2^6} possible choices). As expected, our procedure was always successful in finding a network. An average of 15.8 ± 2.2 hidden units was found, significantly lower than the one obtained from the GM solution: 32. Nevertheless, generalization was completely inexistent for these functions. Indeed, using our learning procedure on only half of the 64 input patterns, the network obtained was able to correctly classify only half of the remaining 32 unlearned patterns. This is just as good as a random guess. Note, however, that by choosing a function at random, we have a very high probability of obtaining a function with no regularities and hence no "predictability." Moreover, we have not being able to obtain any data compression for these functions. Quite on the contrary, compared with the 64 bits needed to tabulate the function, we needed 226 ± 36 bits to store the network (2 bits per connections and $\log_2 \mathrm{K}$ bits per threshold are needed). This feature is also present when one is trying to compress a random file with a standard

data compression technique like Huffman coding.

Our results change drastically when symmetric and regular tasks are used. For instance, when running the parity-K problem on the full set of input/output patterns, we always found the optimal solution of K parallel hidden units (planes) with the normal vector of (1, 1, ..., 1) (or rotated) plane. Moreover, generalization was very good: learning only half the patterns with our procedure yields a network that correctly classify almost all the remaining patterns (we have obtained more than 85% success with parity-6). This is consistent with the fact that these functions are regular and "predictable." In fact, they are made of strongly correlated patterns: changing a bit in the input always changes the output bit.

Now we want to find out if our learning procedure is capable of generating networks with good generalization ability when the function to be learned is somewhere in between these two previous extreme cases — that is, the patterns to be learned are correlated, but not perfectly. Hence, we first need some way to control the correlations between the patterns in a function. For this, consider the set of all Boolean functions defined on N_{inp} input units. Each function can be viewed as a possible configuration of the $2^{N_{\text{inp}}}$ binary variables $\{n_i\}$ (logical variables, spins) defined on the vertices of the unit hypercube of dimension N_{inp}. We want to control to what extent the color of a vertex depends on the color of the nearest-neighbor vertices. This can be achieved by introducing an *energy* for each Boolean function as

$$E(\{n_i\}) = -J \sum_{<i,j>} (2n_i - 1)(2n_j - 1), \tag{4.1}$$

which is the Ising Hamiltonian. The sum runs over all $<i,j>$ pairs of vertices connected by an edge of the hypercube. We will then choose the Boolean functions according to the Boltzmann distribution

$$\text{Prob}(\{n_i\}) \sim \exp(-E(\{n_i\}) \tag{4.2}$$

This distribution gives a higher probability for functions having large correlations (low-energy configurations) than for functions having small correlations (high-energy configurations). $J > 0$ favors positive correlations like in a ferromagnet whereas $J < 0$ favors negative correlations like in an antiferromagnet (parity-like function). $|J|$ plays the role of an inverse temperature. For $|J| = 0$, all Boolean functions have an equal probability of being chosen and hence most probably one picks up a random one. On the oposite limit $|J| \to \infty$, one obtains with probability one either a ferromagnetic (all vertices have the same color, separable case) structure if $J > 0$ or an antiferromagnetic (parity function) structure on the hypercube if $J < 0$. For finite, nonzero J, we will have a certain probability of choosing a function in which the patterns are partially correlated. Hence, J is a parameter controlling the correlations between the input patterns.

Our results are summarized in figure 4 for Boolean functions defined on six input units. For each of these points, we have generated by a Monte Carlo method [9] 100 different Boolean functions and applied our learning

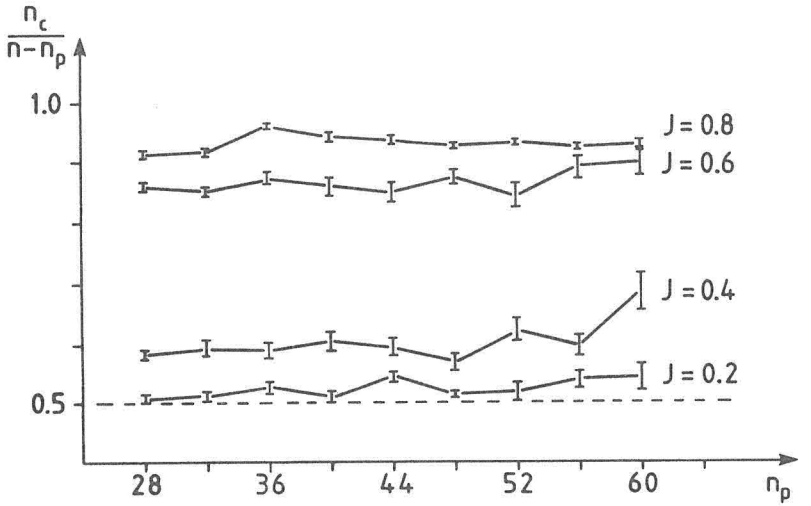

Figure 4: Generalization ability in networks with six input units. At each temperature we have generated by a Monte Carlo method 100 different Boolean functions and applied our learning procedure to each. From the total of 64 patterns, N_{patt} randomly chosen patterns have been learned, then the remaining patterns have been presented to the net. In total N_c such inputs have been correctly classified and $G = N_c/N - N_{\text{patt}}$ has been calculated. A random guess corresponds to a value of $G = 0.5$.

procedure to each. For thermalization, we used 10^6 Monte Carlo (MC) steps, followed by 10^4 MC steps between each sample taken.

It is pretty clear that our learning procedure generates networks with good generalization ability for Boolean functions chosen at low temperature. For example, we see that we have roughly 80% success for $J = -0.8$ and $N_{\text{patt}} = 32$. At these temperatures, the regular (parity) structure breaks into some large clusters requiring only a few additional hidden planes, as seen in figure 5. Presenting a few points from each cluster will create the desired architecture and the system will still be "predictable." However, it is clear that the generalization ability decreases drastically at higher temperature as the patterns become less correlated. In large networks, this sudden deterioration is triggered by a second-order phase transition of the underlying Ising model. Moreover, we have observed also a reduction in the average number of hidden units $< N_h >$ needed to learn the task as $|J|$ is increased, as shown by figure 5.

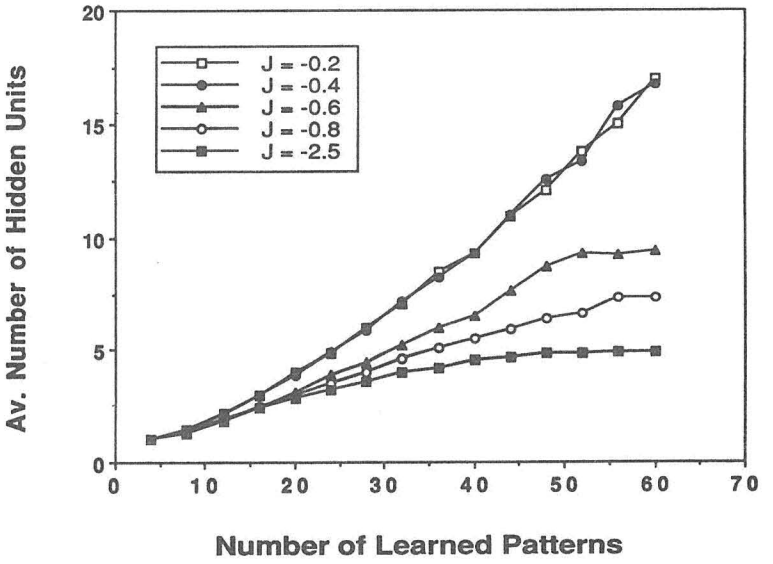

Figure 5: Average number of hidden units constructed by the greedy alogorithm in order to learn the same examples as in figure 4.

5. Conclusion

The main message of this paper is the following: since learning cannot be a generically hard optimization problem, one has to think about learning in a much broader context. Instead of trying to load a given task into a given architecture, as done in other approaches [1–3], we try to synthetize a minimal network in order to learn fast and obtain good generalization properties. We have shown that it is possible to define learning procedures that will find, in polynomial time, networks that have good generalization properties for "regular" tasks (when the patterns to be learned are correlated). Using the notion of regular partitionings, we have presented one such learning procedure, although many more variants are possible.

One obvious deficiency of our actual implementation is that the amount of computation necessary to find a network increases as the number of *possible* input patterns $N = 2^K$ and not only with the *actual* number of patterns to be learned N_{patt}, which in most practical problems is much smaller. As N_{patt}/N decreases, so does the efficiency of our greedy algorithm. It nevertheless remains polynomial as long as this fraction is finite. Obviously, we now need an algorithm that scales as N_{patt} in order to attack some real world pattern recognition tasks. Only then could one possibly fulfill some of the ambitious claims made recently.

Note added. After submission of this paper for publication, we obtained a preprint of an independent work by M. Mézard and J.P. Nadal (ENS preprint, 1989) adopting the same philosophy of constructing a network dur-

ing learning. Their algorithm is quite different from ours (for example, they might need to construct several layers of hidden units before the projection of the task on the last hidden layer is linearly separable) but their results concerning generalization ability are similar to ours. Their algorithm has the advantage, however, of depending mainly on the number of actual examples.

References

[1] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*, Vols. 1, 2 (Bradford Books, MIT Press, Cambridge, 1986).

[2] M.L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, Cambridge, 1969).

[3] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Nature*, **323** (1986) 533–536.

[4] S. Judd, *Proceedings of the IEEE First Conference on Neural Networks*, Vol. II, San Diego, IEEE Cat. No. 87TH0191-7 (IEEE, 1987) 685–692.

[5] J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, and L. Jackel, *Complex Systems*, **1** (1987) 877–922.

[6] G. Tesauro and B. Janssens, *Complex Systems*, **2** (1988) 39–44.

[7] W.V. Quine, *American Mathematical Monthly*, **62** (1955) 627–631.

[8] E.J. McCluskey Jr, *Bell System Technical Journal*, **35** (1956) 1417–1444.

[9] K. Binder and D.W. Heermann, *Monte Carlo Simulation in Statistical Physics* (Springer-Verlag, Heidelberg, 1988).